INTERACTIVE ANALYSIS OF GRAPH AND TIME-SERIES DATA: ENABLING TECHNOLOGIES AND SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Dezhi Fang

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science in the
College of Computing

Georgia Institute of Technology

May 2018

INTERACTIVE ANALYSIS OF GRAPH AND TIME-SERIES DATA: ENABLING TECHNOLOGIES AND SYSTEMS

Approved by:

Dr. Duen Horng Chau
College of Computing
Georgia Institute of Technology

Dr. Oded Green
College of Computing
Georgia Institute of Technology

Date Approved: May 3, 2018

What I cannot create,

I do not understand.

— Richard Feynman

For my parents,

Qun and Jie.

ABSTRACT

Massive amounts of data are being generated every day. While data become more universally accessible, they are also becoming increasingly more complex. With the rise of social networks and mobile sensors, graph and time series data are gaining interest in the research community. However, because of the complexity of such data, making sense of them is still a great challenge. In this thesis, we investigate techniques and systems that enable interactive visual analysis of graph and time series data. We focus on (1) technologies that enable scalable data mining algorithms on a single machine, (2) web-based large scale visualization systems, and (3) these new tools' application on two scenarios: mobile healthcare sensor data, and I/O latency data for computer clusters. The topics of this thesis lie in the intersection of data-mining, human-computer interaction, and database systems. We believe our work will inspire more innovations for interactive interpretation of big data, and human-in-the-loop data analytics systems.

ACKNOWLEDGEMENTS

Polo, I have been unbelievably lucky to have worked with you in my undergraduate years. Thank you, for showing me the world of academia, and trusting me when I had few credentials. Choosing you as advisor, and Polo Club of Data Science as the place to spend most of my time at Georgia Tech is one of the most blessed choices I have ever made in my life.

Oded, thank you for trusting me in leading the software component in the student cluster competition. It has been a tremendous honor to work with you. Will Powell, as well as Petros Eskinder, Alok Tripathy and the rest of the student cluster team, thank you for a wonderful year of cluster-building and compiler-compiling. Your dedication and leadership inspire me.

Dr. Mark Riedl and Chris Prudy, thank you for a year of collaboration, and fascinating work on entertainment intelligence.

My fellow students in Polo Club of Data Science, it has been a pleasure to work with you, and I wish all of you the best in your future careers. To Fred Hohman, Dr. Robert Pienta, and Shang-Tse Chen, I will miss sitting next to you and your inspiring discussions.

Dr. Tobias Wilson-Bates, thank you for your help in my proposal and thesis, as well as your fascinating feedback about my project.

Horus (吴牧宸), thank you for your love and support.

My parents, Qun Fang (方群) and Jie Zhang (张洁), I have never been one for words, but please know that I miss you deeply, and cannot thank you enough for your sacrifices and love that made me who I am today.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Massive amounts of data are being generated every day. However, while the datasets are becoming more available, so are they becoming more complicated. Such complexity comes from various sources, such as structure, dimensionality, and quantity. With the seemingly infinite influx of data, the interpretation of them is even more critical to decision-making processes.

Existing tools for interactive analytics are often either flexible but more suitable for smaller amount o, or scalable but do not emphasise on the visualization component. This thesis focus on enabling technologies and systems that help the interpretation of complex datasets via interactive visual analytics. We present several new techniques for interactive analytics and novel applications of such technologies on graph and time series data.

1.1 Why Interactive Analysis for Time Series and Graph Data?

Recently, with the increasing popularity of social networks and mobile sensors, graph and time series data are gaining interest in the research community. However, due to their complexity, interpreting such datasets is still challenging. Many research projects today focus on one aspect of the problem. They are either focusing on large scale data mining systems, such as Spark [1], or are primarily focused on presentation of data, such as VEGA [2]. While successful in each of its domain, they are not built with the consideration of bridging user-facing and data-facing systems.

There have been many attempts at interactive exploration of graph data, such as Gephi [3] and Cytoscape [4], however, working with large-scale datasets, such as in the context of Gephi, million-node graphs, was never the intention of such tools.

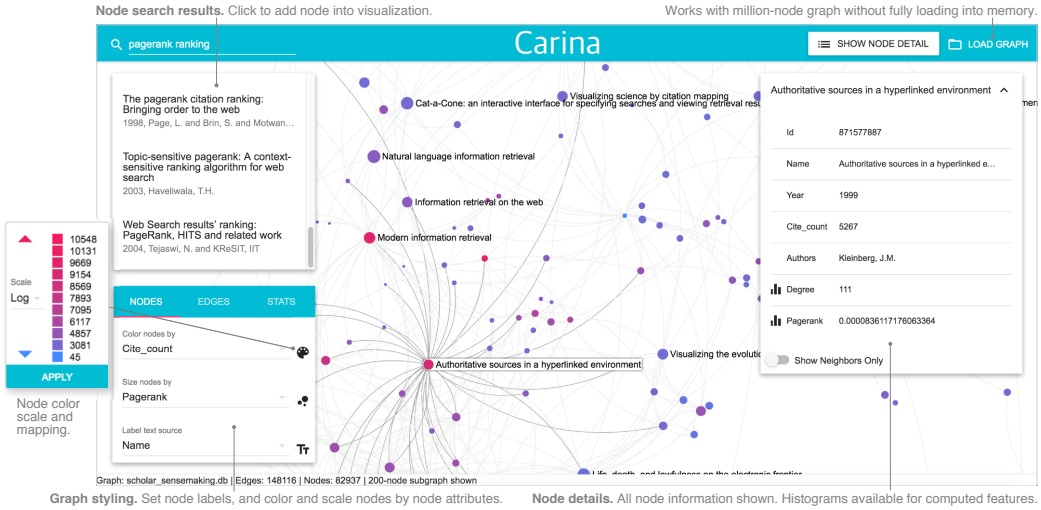

Figure 1.1: Large Scale Visualization on the Web Platform.

With the vast abundance of data, these systems are showing their limits.

1.2 Thesis Overview & Main Ideas

In this dissertation, we study enabling technologies and systems for interactive visual analysis of graph and time series. We divide our investigation into multiple interrelated chapters, some focus on techniques and some on systems.

1.2.1 Technique: Efficient Data-Processing on a Single Machine

In this chapter, we talk about current challenges in the efficiencies of data mining tools. We especially pay attention to the systems that aim to work on a small cluster or a single PC, since those are most frequently used for interactive exploration. We look at the overhead of cluster-oriented architectures popular in today's data mining pipelines, and introduce methods to scale the capability of a single machine. These methods are essentially specialized versions of general-purpose systems: they rely on the underneath dataset or problem to be formulated in a way that's friendly for sequential I/O access [5]. In those scenarios, according to the roofline model [6], places the bottleneck on processor power rather than random access memory,

Figure 1.2: Exploratory Time Series Visualization for Mobile Sensor Data

which is relatively plentiful on a single PC. These methods can be embedded in other applications to support exploratory analytics.

1.2.2 Technique: Large Scale Visualization on the Web Platform

In this chapter, we talk about accessible methods to scale up visualization systems. In particular, we look to exploit the growing adoption of the web platform to aid visualization [7]. Web has become the de facto universal operating system. Due to the popularity of the platform, plenty of efforts has gone into optimizing the performance of modern and complicated applications. We argue that the advance of technologies such as WebGL [8] and V8 JIT engine[9] has made it possible to develop high performance visualization systems based on those technologies. We showcase the capability of Web platform with the work in [7], where a large-graph visualization program was developed with web technologies that supports datasets larger than its predecessors. A number of techniques found in Computer Graphics are applied in order to optimize the performance. Further, because of the use of Web Technologies, the application is naturally portable, either available directly in a browser, or through a wrapper such as Electron.js (Electron).

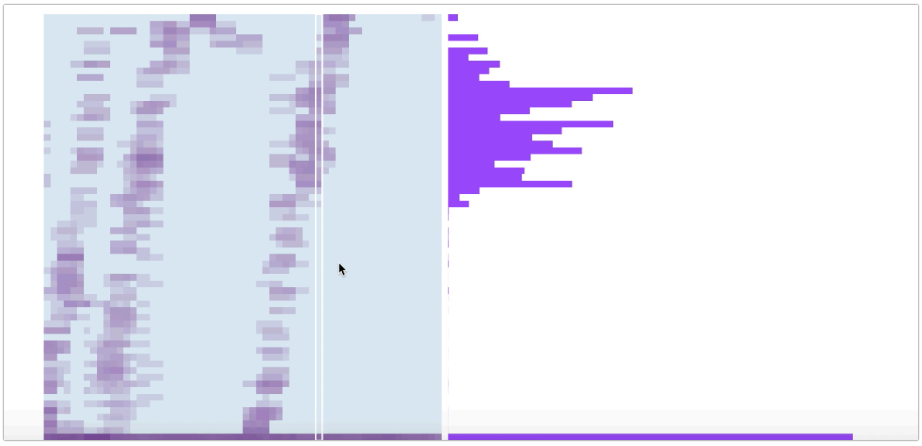

Figure 1.3: Interactive Heat Map Analysis for System Latency

1.2.3 Application: Exploratory Time Series Visualization for Mobile Sensor Data

In this chapter, we take the results of the first two approaches, and construct systems that are both scalable and interactive explorable. Due to the nature of interactive exploration, these systems need to be able to complete analysis in a timely fashion, while large datasets often go against that goal. We discuss methods we used to scale those system up for interactive exploration, such as caching, proper choice of components including data storage, overall architecture of distributed systems for interactive analysis, as well as optimizations done in the frontend.

1.2.4 Application: Interactive Heat Map Analysis for System Latency

In this chapter, we look at another novel application of interactive analysis: I/O latency heat maps. We use the techniques described in this thesis and apply them to performance monitoring. We present a use case where the aggregation of large amount of time series data can provide context to ongoing performance incidents, by intelligently summarizing high-granularity latency information across a cluster in a single heat map. We also present performance optimization techniques applied to the agents collecting the latency information in order to minimize the performance impact of the measurements. We use `eBPF`, a technique that shifts the aggregation

of data directly into the kernel, combined with the interactive analysis, to provide context previously difficult to gather.

1.3 Impact

- The *mHealth Discovery Dashboard* has been used to analyze two distinct datasets from studies of smoking secession, and has generated insights about the behaviors around smoking relapses.
- *ARGO*, the successor of *Carina*, has been deployed to *CSE 6242: Data and Visual Analytics*, a popular course in Georgia Tech for two semesters, used by a total of more than 600 students.

CHAPTER 2
LITERATURE REVIEW

As large datasets and machine learning models become increasingly common, they are also increasingly complicated. A modern data-mining pipeline often consists of many parts [10]. While many are individually well-understood, when composed together, the systems and datasets are often viewed as black boxes [11]. Thus, by bridging data mining systems with intelligent interfaces, we can help people more easily understand relations among entities, and can help reveal useful insights hidden in the data and models intuitively. But, existing visualization systems face multiple technical, visual, and scalability challenges.

Visual Scalability Challenge. For visualization systems that follow the conventional approach of visualizing the entire dataset (e.g., Cytoscape [4]), it is very common to the have noise in the data overwhelm any interpretation. For example, for million-node graphs, such visualizations generate "hairballs" with extreme edge crossings [12], overwhelming human perception and impeding understanding.

Data Scalability Challenge. In most existing systems, especially graph visualization systems, a dataset must first be fully loaded into memory. The available RAM becomes a barrier for analyzing larger datasets with GBs of data and more For example, the popular Gephi [3] system runs out of RAM when trying to load the LiveJournal social graph with 69M edges [13].

Information Summarization Challenge. Similar to the Data and Visual Scalability challenges, even when a system can handle large amounts of data as well as presentation of such data, the value of the visualization can be diminished by the large amount of data available. Researches have shown that human can hold a small num-

ber of items in working memory [14], a number much smaller than the vast amount available to machines. Thus, for a large-scale visualization system to be effective, *insights* must be extracted for presentation [15]. For this reason, previous visualization systems such as TimeStitch often focus on high-level pattern summarization[16]. both low-level and high-level exploration tools for visualizing raw data, interactively inspecting it to formulate hypotheses, and discovering trends and patterns [17]. To address this conflict, there must be an intelligent mix of both low-level and high-level exploration in efficient visualization systems.

System Usability Challenge. Most existing visualization tools are built primarily for desktop use, precluding analysis in mobile environments, which are increasingly common (e.g., DARPA GUARD DOG mobile graph analytics [18]). For instance, two of the most popular graph visualization tools, Gephi and Cytoscape are built using Java and run only on desktop computers. A large number of data-mining tools, e.g., Spark [1], requires a cluster to perform well. This restriction further limits the availability of systems that are considered "Human-in-the-loop" [19], as setting up a cluster is often too complicated for a visualization system to integrate with. However, researches have shown that in many cases, a simple PC or even laptop can perform as well as or even better than a moderately scaled cluster [20] [5].

CHAPTER 3
TECHNIQUE: SCALABLE DATA MINING SYSTEMS ON A SINGLE MACHINE

One of the challenges of interactive exploration systems is the performance of their data mining components. Because interactive visualizations often require low latency feedback to user actions [21], the performance of these applications can have significant impact on the application's overall effectiveness. Approaches such as offloading computation to a cluster have been explored, but are often undesirable due to the inherent latency with network requests, especially when dealing with large amounts of data.

In this chapter, we look at approaches that scales data mining on a single system. Extensions of the techniques are being used in applications such as ARGO [7], to enhance the experience of large scale visualization systems.

3.1 Introduction

To process data that do not fit in RAM, conventional wisdom would suggest using distributed approaches. However, recent research has demonstrated virtual memory's strong potential in scaling up graph mining algorithms on a single machine.

We propose to use a similar approach for general machine learning.

We contribute:

1. our latest finding that memory mapping is also a feasible technique for scaling up general machine learning algorithms like logistic regression and k-means, when data fits in or exceeds RAM (we tested datasets up to 190GB);

2. an approach, called M3, that enables existing machine learning algorithms to

work with out-of-core datasets through memory mapping, achieving a speed that is significantly faster than a 4-instance Spark cluster, and comparable to an 8-instance cluster.

3.2 Motivation

Leveraging virtual memory to extend algorithms for out-of-core data has received increasing attention in data analytics communities. Recent research demonstrated virtual memory's strong potential to scale up graph algorithms on a single PC [20, 22]. Available on almost all modern platforms, virtual memory based approaches are straight forward to implement and to use, and can handle graphs with as many as 6 billion edges [22]. Some single-thread implementations on a PC can even outperform popular distributed systems like Spark (128 cores) [20]. Memory mapping a dataset into a machine's virtual memory space allows the dataset to be treated identically as an in-memory dataset. The algorithm developer no longer needs to explicitly determine how to partition the (large) dataset, nor manage which partitions should be loaded into RAM, or unloaded from it. The OS performs similar actions on the developer's behalf, through paging the dataset in and out of RAM, via highly optimized OS-level operations.

3.3 Scaling Up using M3

As existing works focused on graph algorithms such as PageRank and finding connected components, we are investigating whether a similar virtual memory based approach can generalize to machine learning algorithms at large.

Inspired by prior works on graph computation, our M3[1] approach uses memory mapping to amplify a single machine's capability to process large amounts of data for machine learning algorithms. As memory mapping a dataset allows it to be

[1]M3 stands for Machine Learning via Memory Mapping.

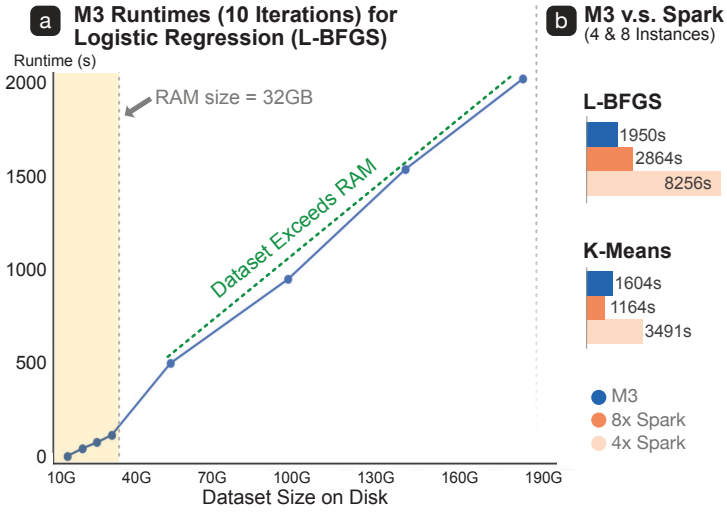

Figure 3.1: a: M3 runtime scales linearly with data size, when data fits in or exceeds RAM. b: M3's speed (one PC) comparable to 8-instance Spark (orange), and significantly faster than 4-instance Spark (light orange).

Original	**M3**
`Mat data;`	`double *m = mmapAlloc(file, rows * cols);` `Mat data(m, rows, cols);`

Table 3.1: M3 introduces minimal changes to code originally using in-memory data structure, enabling it to work with much larger memory-mapped data.

treated identically as an in-memory dataset, M3 is a transparent scale-up strategy that developers can easily apply, requiring minimal modifications to existing code. For example, Table 3.1 shows that with only minimal code changes and a trivial helper function, existing algorithm implementation can easily handle much larger, memory-mapped datasets.

Modern 64bit machines have address spaces large enough to fit large datasets into (up to 1024PB). Because the operating system has access to a variety of internal statistics on how the mapped data is being used, the access to such data can be further optimized by the operating system via methods including least recent used caching and read-ahead to achieve efficiency [23].

To test the feasibility of M3, we minimally modified mlpack, an efficient machine learning library written in C++ [24]. Memory mapping can be easily applied to other

languages and algorithm libraries.

3.4 Experiments

Our current evaluation focuses on: (1) understanding of how M3 scales with increasing data sizes; and (2) how M3 compares with distributed systems such as Spark, as prior work suggested the possibility that a single machine can outperform a computer cluster [20].

3.4.1 Experiment Setup.

All tests with M3 are conducted on a desktop computer with Intel i7-4770K quad-core CPU at 3.50GHz (8 hyperthreads), 4×8GB RAM, 1TB SSD of OCZ RevoDrive 350. We used Amazon EC2 `m3.2xlarge` instances for Spark experiments. Each instance has 8 vCPUs (hyperthreads of an Intel Xeon core) with 30GB memory and 2×80GB SSDs. The Spark clusters are created by Amazon Elastic MapReduce and the datasets are stored on the cluster's HDFS.

3.4.2 Dataset.

We used the *Infimnist*[2] dataset, an infinite supply of digit images (0–9) derived from the well-known MNIST dataset using pseudo-random deformations and translations. Each image is 28×28 pixel grayscale image (784 features; each image is 6272 bytes). We generated up to 32M images, whose dense data matrix representation contains 23.5 billion entries, amounting to 190GB. Smaller datasets are subsets of the full 32M images.

[2] `http://leon.bottou.org/projects/infimnist`

Setup	L-BFGS	Gradient Descent	Randomized SGD
MMap	5232s	616s	2076s
4xSpark	5500s	605.6s	N/A (Job Failure)
8xSpark	200s	100+s	100+s

Table 3.2: Comparing M3 to Spark

3.4.3 Algorithms Evaluated.

We selected *logistic regression* (L-BFGS for optimization) and k-means, since they are commonly used classification and clustering algorithms.[3]

3.5 Key Findings & Implications

1. M3 scales linearly when data fits in RAM and when out-of-core, for logistic regression (Figure 3.1a). The dotted vertical line in the figure indicates RAM size (32GB). M3's runtime scales linearly both when the dataset fits in RAM (yellow region in Fig. 3.1a), and when it exceeds RAM (green dotted line), at a higher scaling constant, as expected.

Looking at M3's resource utilization, we saw that M3 is I/O bound: disk I/O was 100% utilized while CPU was only utilized at around 13%. This suggests strong potential for M3 reaching even higher speed if we use faster disks, or configurations such as RAID 0.

2. M3's speed (one PC) is comparable to 8-instance Spark and significantly faster than 4-instance Spark for logistic regression (L-BFGS) and k-means (Figure 3.1b). This result echoes prior works focusing on graph computation that suggests cluster may not be necessary for moderately-sized datasets [20, 22]. Our result extends those findings to two general machine learning methods.

For *logistic regression* (with 10 iterations of L-BFGS), M3 is about 30% faster than 8-instance Spark. 4-instance Spark's runtime was 4.2X that of M3. For *k-means*

[3]We are primarily interested in runtimes, so we did not perform image pre-processing.

(10 iterations, 5 clusters), M3 ran at a speed comparable to 8-instance Spark (1.37X), and more than twice as fast as 4-instance Spark.

Certainly, using more Spark instances will increase speed, but that may also incur additional overhead (e.g., communication between nodes). Here, we showed that for moderately-sized datasets, single-machine approaches like M3 can be attractive alternatives to distributed approaches.

3.6 Conclusions

We are taking a first major step in assessing the feasibility of using virtual memory as a fundamental, alternative way to scale up machine learning algorithms. M3 adds an interesting perspective to existing solutions primarily based on distributed systems.

We contribute: (1) our latest finding that memory mapping could become a feasible technique for scaling up general machine learning algorithms when the dataset exceeds RAM; (2) M3, an easy-to-apply approach that enables existing machine learning implementations to work with out-of-core datasets; (3) our observations that M3 on a PC can achieve a speed that is significantly faster than a 4-instance Spark cluster, and comparable to an 8-instance cluster.

CHAPTER 4
TECHNIQUE: LARGE SCALE VISUALIZATION ON THE WEB PLATFORM

In this chapter, we present a scalable, interactive visualization system, called Carina, that helps people explore million-node graphs. By using latest web browser technologies, Carina offers fast graph rendering via WebGL and works across desktop (via Electron) and mobile platforms. Different from most existing graph visualization tools, Carina does not store the full graph in RAM, enabling it to work with graphs with up to 69M edges.

4.1 Introduction

Large graph data have become increasingly common. Visualizing them help people more easily understand relations among entities. But, existing graph visualization systems face multiple technical, visual, and scalability challenges.

Visual Scalability Challenge. Most graph visualization systems follow the convention approach of visualizing the entire graph (e.g., Gephi [3], Cytoscape [4]). For million-node graphs, such visualizations generate "hairballs" with extreme edge crossings [12], overwhelming human perception and impeding understanding.

Data Scalability Challenge. In most existing systems, a graph dataset must first be fully loaded into memory. The available RAM becomes a barrier for analyzing larger graphs with million nodes or more. For example, the popular Gephi system runs out of RAM when trying to load the LiveJournal social graph with 69M edges [13].

Technology Challenge. Most existing visualization tools are built primarily for desktop use, precluding analysis in mobile environments, which are increasingly common

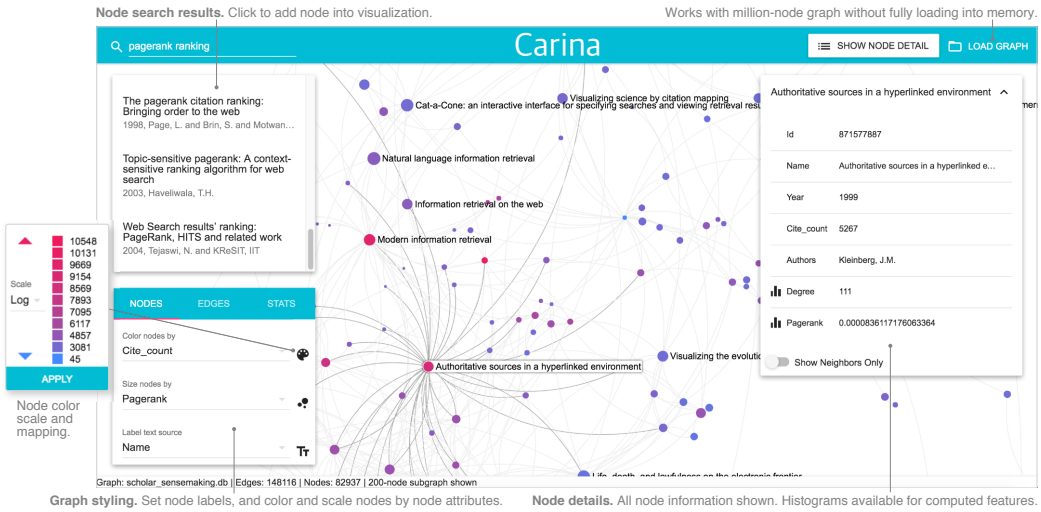

Figure 4.1: Carina visualizing citation network data (83k nodes, 150k edges) [12]. Carina works with out-of-core million-node graphs (up to 69M edges). Carina uses latest web browser technologies for high-performance graph rendering (WebGL), easy cross-platform deployment (Electron), and lightweight, scalable data storage.

(e.g., DARPA GUARD DOG mobile graph analytics [18]). For instance, Gephi and Cytoscape are built using Java and run only on desktop computers.

Our ongoing research to tackle the above challenges:

- We introduce Carina (Fig. 4.1), a graph visualization system developed using the latest web browser technologies, It offers fast graph rendering via WebGL, and lightweight, cross-platform deployment via Electron.
- We demonstrate that Carina works with million-node graphs (up to 69M edges), without requiring them to fit in RAM, unlike most existing graph visualization tools. Carina visualizes user-specified subgraphs (instead of the whole graph), allowing users to focus their exploration on the most relevant graph regions. Carina uses SQLite for storing and querying out-of-core datasets.

4.2 Implementation

High-performance Graph Rendering via WebGL. Many visualization libraries, e.g., d3.js (`https://d3js.org`) and processing.js, render graphs using technologies like *Scal-*

able Vector Graphics (SVG) and HTML Canvas elements. However, their rendering speed (screen refresh rate) begins to deteriorate significantly starting at low hundreds of nodes and edges (`https://github.com/anvaka/graph-drawing-libraries`).

We have identified WebGL as a high-performance graphics technology that has strong potential for fast graph rendering. WebGL uses GPU acceleration, and is supported by all modern web browsers (`http://caniuse.com/#feat=webgl`). Carina leverages WebGL's high-speed graphics capabilities through Three.js, a higher-level library designed to simplify WebGL programming (`https://threejs.org`). Carina adapts many 3D accelerated graphics techniques from WebGL to 2D space for graph visualization. In particular, we use level-of-detail, buffer geometry, and particle systems for fast rendering of nodes and edges.

Visual Scalability on Real Data. To better understand WebGL's rendering scalability, we tested Carina with graphs of varying sizes, on a MacBook Pro laptop (2015 version, i7-4870HQ, 16GB RAM). For the YEAST dataset with 2361 nodes and 7182 edges, user interactions such as panning, zooming and dragging nodes, with a force-directed graph layout algorithm running in the background, achieved a smooth frame rate of 60FPS. The frame rate only starts to drop below 30FPS when the graph size exceeds 20k nodes and 56k edges. We note that when analyzing real-world power-law graphs, we typically would not want to visualize the entire graph (which shows up as a "hairball"). We conducted the above experiment to understand the technological limits.

Cross-platform Integration and Deployment. We design Carina with platform portability in mind, hence our decision to use latest web browser technologies. Carina can be deployed as a desktop application that runs on all popular operating systems (Linux, Windows, Mac), via the Electron framework based on the Chrome browser and Node.js (`http://electron.atom.io/`). Electron packages Carina as self-contained binaries for all platforms without assumptions of the user's run-time environment.

Electron also grants Carina native I/O and high performance Inter-Process Communication (IPC) capabilities. Carina can also be used in any modern browsers on mobile devices and desktops.

Million-node Graph Data Storage and Exploration. Different from most graph visualization tools, Carina does not store the full graph in RAM; only the visualized subgraph is kept in memory, allowing Carina to work with graphs with up to 69M edges [13]. Currently, Carina stores a million-node graph in an out-of-core SQLite database. We chose SQLite for its simplicity, ease of integration, and scalability for up to tens of millions of edges. The user can select and visualize sub-graphs through techniques such as node search (as in Fig. 4.1) or based on graph features (e.g., computed measures like PageRank scores). SQLite can induce subgraphs quickly. For example, it takes only 120ms to induce a 2000-node subgraph (9867 edges) out of the 69M edge LiveJournal graph.

4.3 Conclusion

We are designing and developing Carina, a scalable, interactive visualization system for million-node graph exploration. Using latest web browser technologies, Carina offers multiple advantages over existing graph visualization tools, such as fast graph rendering (via WebGL), easy cross-platform deployment (via Electron), and scalable data storage and exploration of large graphs with up to 69M edges on commodity machines. We plan to conduct user studies to evaluate the usability of Carina, and work with real domain users, such as security analysts at Symantec, to use Carina to help uncover company insider threats lurking in large email graphs and computer communication networks. We believe Carina provides a new, scalable way for practitioners and researchers to explore and visualize large graphs.

CHAPTER 5
APPLICATION: EXPLORATORY TIME SERIES VISUALIZATION FOR MOBILE SENSOR DATA

In this chapter, we apply the interactive analytics approach to mobile sensor time series data. We present Discovery Dashboard, a visual analytics system for exploring large volumes of time series data from mobile medical field studies. Discovery Dashboard offers interactive exploration tools and a data mining motif discovery algorithm to help researchers formulate hypotheses, discover trends and patterns, and ultimately gain a deeper understanding of their data. Discovery Dashboard emphasizes user freedom and flexibility during the data exploration process and enables researchers to do things previously challenging or impossible to do — in the web-browser and in real time. We demonstrate our system visualizing data from a mobile sensor study conducted at the University of Minnesota that included 52 participants who were trying to quit smoking.

5.1 Motivation

When medical researchers conduct mobile sensor field studies, they often collect large amounts of time series data across many participants over prolonged periods of time. When incorporating data science techniques into the healthcare domain, making sense of the data collected from mobile health devices is essential for gaining actionable insights [25].

This volume of data (raw or pre-processed) can be overwhelming to a researcher seeking to gain such insights.

For this reason, previous efforts such as TimeStitch often focus on high-level pattern summarization [16]. However, to test hypotheses and obtain a deep understand-

Figure 5.1: The Discovery Dashboard interface showing data from a mobile sensor study. Each row corresponds to one participant's data. A user-defined motif (for participant 6012) is selected, and the system automatically finds similar motifs across all participants and highlights them in yellow. This particular motif is a recurring pattern for participant 6012, often found near smoking lapses (vertical red dotted lines).

ing of ones data, researchers need both low-level and high-level exploration tools for visualizing raw data, interactively inspecting it to formulate hypotheses, and discovering trends and patterns.

To address both low-level and high-level exploration, we present *Discovery Dashboard*: a visual analytics system that offers intuitive visualization of mobile sensor time series data, supports multiple interaction techniques for data and pattern exploration, and integrates a data mining algorithm for motif discovery.

5.2 Mobile Sensor Dataset

We use data from a four day mobile sensor clinical study conducted at the University of Minnesota. The study aimed to uncover what causes relapse in cigarette smokers attempting to quit smoking. The research included a rich design to capture psychological, behavioral, biological, and physiological data related to stress, withdrawal symptoms, affect, and craving as well as lapse events for cigarette smokers attempting to quit [26]. From the 365MB dataset containing 52 participants, we visualize three

time series (1 Hz) for each participant: inferred stress, physical activity, and heart rate, totaling 4.7M data points.

5.3 Discovery Dashboard System and Design

In Figure 5.1, Discovery Dashboard visualizes raw time series data of 52 participants, each represented by a single row, consisting of 24-hour blocks. Researchers can (1) align the time series by first smoking lapse, (2) filter participants by name, number of lapses, and the day of their first lapse, and (3) search for user-defined time series motifs (shown in Figure 5.2).

Discovery Dashboard is a web-based visualization system that can be run using any modern browser. However, using the web as a platform for making sense of large volume data presented interesting computational challenges. Below we describe some of our design decisions that enable the real time interactive experience in Discovery Dashboard.

5.3.1 Scalability for Interactive Exploration

To support interactive exploration on data with high resolution, Discovery Dashboard needs to scale to large datasets; therefore we introduced multiple caching layers to achieve such scalability. Discovery Dashboard uses a relational database (SQLite) for storing the raw data and a key-value store (Redis) for caching resampled data and motif results. Time series data are pre-processed and manipulated with the Pandas package in Python and motif data are calculated using the Symbolic Aggregate approXimation (SAX) algorithm [27], a popular time series transformation method, written in Java. These services communicate via gRPC, a high performance remote procedure call (RPC) framework that transmits data using Google's Protocol Buffers. Calculated data are then transmitted to the client through WebSockets, transformed with D3.js, and rendered to the browser with React.js for maximum client perfor-

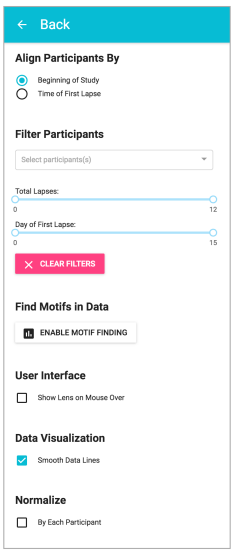

Figure 5.2: The Discovery Dashboard contains a number of options that are accessible from the "Analyze Particpants" button. Researchers can (1) align the data chronologically or by the first smoking lapse, (2) filter participants by name, number of lapses, and the day of their first lapse, and (3) search for time series motifs.

mance.

5.3.2 Motif Finding Algorithm

Finding motifs in time series data can be challenging, especially noisy data such as those collected from mobile sensor hardware. Discovery Dashboard uses the symbolic time series representation SAX algorithm [27] for its high performance when detecting latent patterns in noisy time series data. For example, in Figure 5.1, the zoomed in region under participant 6012 is used as motif query: similar patterns (highlighted in yellow) are found by SAX in the time series of participant 6012 and 6013, based on the patterns' similar "shapes" to the initial query motif, rather than their absolute temporal values.

5.3.3 mHealth Time Series Alignment

Participants' time series data visualized in Figure 5.1 are aligned by the experiment start date, a natural alignment helpful for understanding the overall patterns across participants. However, aligning the visualization by other user-defined events, such as smoking lapses (vertical red dotted lines in Figure 5.1), can also help gain insights. For example, by aligning the data by first smoking lapses, we can more easily compare the different patterns that participants exhibited right before and after lapses.

CHAPTER 6
APPLICATION: INTERACTIVE HEAT MAP ANALYSIS FOR SYSTEM LATENCY

Performance monitoring is an important aspect of modern application development. For example, it's been long known that the speed of a website is strongly correlated with customer loyalty[28]. With more and more companies moving their applications into the cloud, performance can also directly influence capacity planning and thereby the operational cost. The ability to derive complicated insights, often spanning different services in multiple hosts, is therefore critical. In this chapter, we investigate applying interactive analytics to monitoring I/O latency distribution.

6.1 Introduction

Many performance related metrics, such as TCP latency, block device access latency, and file system caches are traditionally only available from an in-kernel context. Monitoring them from tools such as `strace` often incurs inhibiting costs that prevent those tools from being deployed to production systems. With the recent addition of eBPF to Linux kernels, however, we're able to monitor such metrics with very low overhead. We propose a performance monitoring tool, `eBPF` Heat Maps, that uses eBPF to gather latency of different system calls, aggregate the information across a cluster, and then present the information in heat maps, indicating different performance aspects in a cluster.

6.2 Related Work

6.2.1 extended Berkeley Packet Filters

extended Berkeley Packet Filters, or eBPF, is an in-kernel virtual machine. Its earlier form, BPF, was originally used for efficient packet filtering in `tcpdump`[29]. Since packets often come in a fast rate, packet filtering needs to be done in a high-performance way that doesn't incur too much latency. BPF is then built to attach user supplied code to the kernel network stack, so that packet filtering does not require context switches to user mode. eBPF, an enhanced version of BPF, is able to be attached to `kprobe`s. This ability enabled users to live-instrument a kernel in a way that's highly efficient and never crashes[30]. Since then, a lot of interest on eBPF is gathered in the observability community. For example, one can measure the latency of block device IO by simply measuring the time between `blk_account_io_start` and `blk_account_io_completion` kernel events.

6.2.2 `strace`

`strace` can be used to monitor and trace system calls in Linux. It is very useful in debugging performance anomalies. However, running an application with `strace` incurs performance-inhibiting overhead, making it impossible to be deployed in production[31].

6.2.3 DTrace

DTrace is a powerful tracing tool originated from Solaris[32]. There is a large collection of DTrace scripts available, many targeting at latency histograms. DTrace is not yet available in Linux. With the rise of tools such as `eBPF` and *SystemTap*, the effort of tracing community is unlikely to be concentrated on DTrace, making it less likely to be future-proof. However, it is entirely possible that DTrace can be used

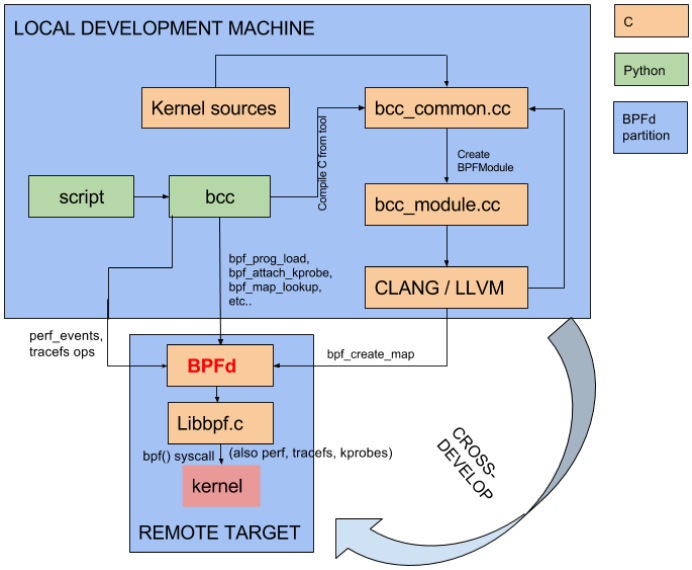

Figure 6.1: Architecture diagram of `BPFd` with `bcc`.

as a front-end to `eBPF` or `perf_events`, enabling Linux performance engineers to use the abundance of DTrace scripts available.

6.2.4 `perf_event`

`perf_event` is a sample profiling tool available in linux. Many performance insights can be derived from `perf_event`, such as CPU saturation and flame graphs. However, `perf_event` is more focused on profiling, and is less flexible than `eBPF` in certain ways. It's most significant limitation perhaps comes from the cloud, as CPU-level performance events are often not available in cloud platforms[33].

6.3 `eBPF` Heat Maps

We built a tool, `eBPF` Heat Maps, that utilizes `eBPF` to gather performance-related latency information, and then present an aggregated visualization in a web dashboard. Here, we describe the different components of the project.

Table 6.1: Examples of metrics that can be monitored with kernel events and eBPF

metrics	tracing start event	tracing end event
Block IO	blk_account_io_start	blk_account_io_completion
TCP Latency	tcp_v[4,6]_connect	tcp_rcv_state_process
Queuing Latency	enqueue_task_*	finish_task_switch

6.3.1 BPFd

Most of eBPF related work is done with the help of bcc, bpf compiler collection [34]. However, bcc has a hard dependency on linux kernel source, as well as the llvm toolchain. These dependencies make it very difficult for eBPF based tools to be deployed to production, as both dependencies are non-trivial. Moreover, these dependencies make deployments on ARM based platforms next to impossible. There alternatives to the bcc approach, as used in BPFd, is a client-server architecture [35]. In this approach, the application server does not need to host the compiler suite, but instead the bpf programs are compiled (or cross-compiled) on a development machine. The application server only needs to host a small agent that loads bpf programs as needed. The architecture of BPFd is shown in Figure 6.1.

6.3.2 BPF Payload

In Table 6.1, we list a few example performance metrics that can be traced with eBPF. Here, we are only listing static tracing points available in kernel. We can also modify the payload to monitor any user-defined tracing points (USDT) such as those available in Node.js and MySQL [36]. With uprobe and dynamic user space tracing, we will be able to trace any function calls. Throughout the course of the project, we will identify more metrics that can be monitored through this method, and present the user with a wide variety of choices.

6.3.3 Collection Agent and Aggregation Server

In order to collect the captured histograms, we deploy a collection agent on each of the instances in a cluster. The collection agent is written with performance consideration in mind, such that it wouldn't impact the application performance by running. The agent is single-threaded and event-driven, written with the asynchronous I/O library, Twisted, to minimize the footprint.

Since we're measuring I/O intensive application, CPU saturation is rarely a concern. Because all aggregation is done in the kernel, the agent would only need to ship an aggregated histogram to the collection server. Furthermore, these updates are sent with Protocol Buffers, which offers a compact size for sparse objects that are often seen in latency heatmaps. Each update message is also compressed with GZIP. In each update interval, the network traffic is often below 1kb.

The aggregation server is written with Flask, and paired with ElasticSearch, a popular search engine often used to analyze logging information. Each update message is indexed and stored in an ElastiSearch document. ElasticSearch enables easy querying for update messages. When the user requests to view a heatmap, the aggregation server will retreve each individual messages according to filters generated by the request. These messages are again aggregated for a cluster, and presented to the user.

6.3.4 Latency Heat Maps

Heat maps are visualizations often used to present information in three dimensions in a 2D figure. Latency heat maps are used to demonstrate the live operational status of a system [37]. An example of latency heat map (from Datadog [38]) can be seen in Figure 6.2. Here, we can see that unlike a histogram of latency, we're able to see baseline information as well as the general trend of system performance. We define the three dimensions of data we're collecting as following:

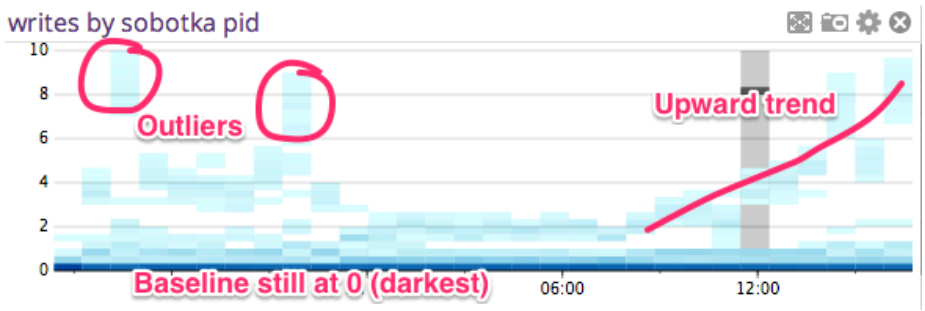

Figure 6.2: An example latency heat map from Datadog.

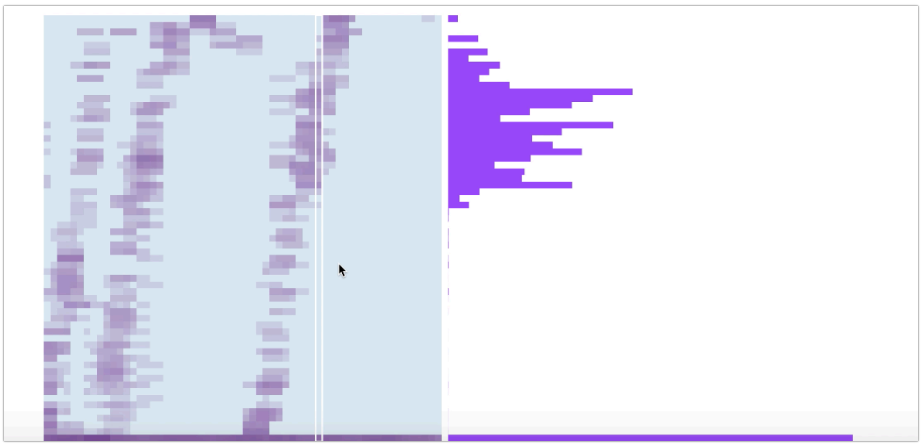

Figure 6.3: The full interface of the eBPF Heat Map

1. Latency
2. Count
3. Timestamp

In addition, since one column of the heat map is effectively the histogram of that specif time slice, we also show the histogram as the user moves the cursor over the different columns, as shown in figure 6.3.

6.3.5 Overall Application Architecture

An overall demonstration of the application architecture can be found in Figure 6.4.

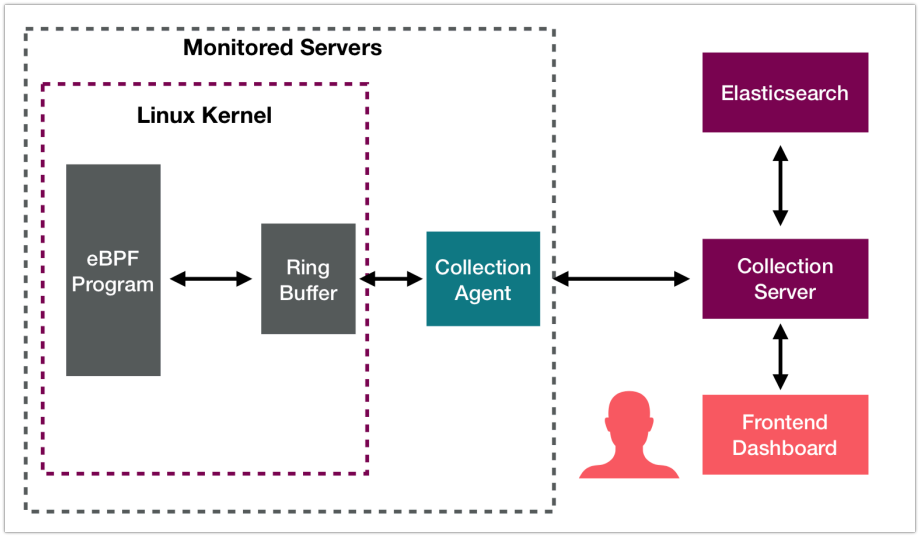

Figure 6.4: Overall Application Architecture

6.4 Experiments

In a case study, we compare the performance of `npm` and `yarn`, two package managers for JavaScript applications. Specifically, we trace the latency distribution of block device I/O during `npm install` and `yarn install`. The experiments are conducted on a 1TB, 5400rpm HDD as well as on a 500GB SSD. We chose a JavaScript application with 66 dependencies, and a total size of 740MB in `node_modules` after all the transitive dependencies are installed. Installing JavaScript is especially sensitive to I/O latency due to its frequent small writes to disk content. As seen in Figure 6.5, SSD offers significantly better performance for both `npm` and `yarn`. While the performance for both `npm` and `yarn` are worse on an HDD, `yarn`'s significantly better cache utilization strategy is able to make more efficient use of the disk cache, and finish the installation in both a shorter time and a more concentrated I/O latency distribution.

6.5 Conclusions

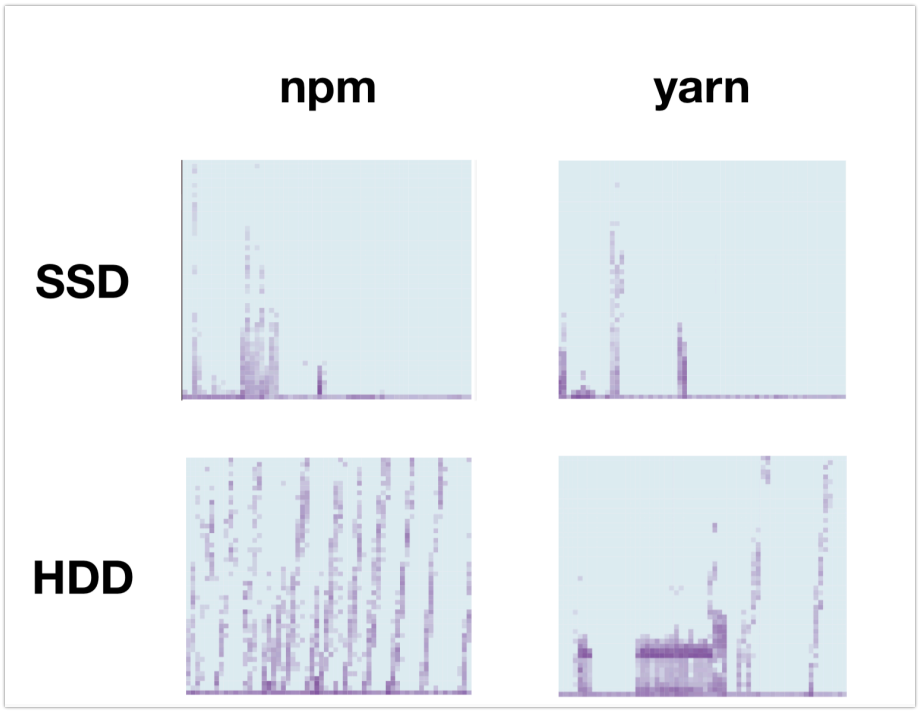

Figure 6.5: A matrix of (SSD, HDD) x (`yarn`, `npm`)

CHAPTER 7
CONCLUSIONS

The abundance of graph and time series data presents many new opportunities for analytics systems, but also present challenges for interpretation due to their complexity. In this thesis, we advocate for a visual and interactive approach to graph and time series analytics systems. We contribute: (1) technologies that enable scalable data mining algorithms on a single machine, (2) web-based large scale visualization systems, , and (3) these new tools' application on two scenarios: mobile healthcare sensor data, and I/O latency data for computer clusters. We believe the work presented in this thesis will inspire more innovations for interactive interpretation of big data, and human-in-the-loop data analytics systems.

REFERENCES

[1] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, "Apache Spark: A unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[2] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-lite: A grammar of interactive graphics," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 341–350, 2017.

[3] M. Bastian, S. Heymann, M. Jacomy, *et al.*, "Gephi: an open source software for exploring and manipulating networks.," *Icwsm*, vol. 8, pp. 361–362, 2009.

[4] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker, "Cytoscape: a software environment for integrated models of biomolecular interaction networks," *Genome research*, vol. 13, no. 11, pp. 2498–2504, 2003.

[5] D. Fang and D. H. Chau, "M3: Scaling Up Machine Learning via Memory Mapping," in *Proceedings of the 2016 International Conference on Management of Data*, ACM, 2016.

[6] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures," Tech. Rep., 2009.

[7] D. Fang, M. Keezer, J. Williams, K. Kulkarni, R. Pienta, and D. H. Chau, "Carina: Interactive million-node graph visualization using web browser technologies," in *Proceedings of the 26th International Conference on World Wide Web Companion*, International World Wide Web Conferences Steering Committee, 2017, pp. 775–776.

[8] "Webgl implementations," in *WebGL Insights*, A K Peters/CRC Press, 2015, pp. 1–2.

[9] D. Botcharnikov, "Approaches to optimizing v8 javascript engine," *Proceedings of the Institute for System Programming of the RAS*, vol. 27, no. 6, pp. 21–32, 2015.

[10] R. Barga, V. Fontama, and W. H. Tok, *Predictive analytics with Microsoft Azure machine learning (2nd Edition)*. Apress, 2015.

[11] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?": Explaining the Predictions of Any Classifier," in *KDD*, ACM, 2016.

[12] D. H. Chau, A. Kittur, J. I. Hong, and C. Faloutsos, "Apolo: making sense of large network data by combining rich user interaction and machine learning," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2011, pp. 167–176.

[13] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford Large Network Dataset Collection*, `http://snap.stanford.edu/data`, 2014.

[14] A. D. Baddeley, A. D. Baddeley, and A. Braddlely, *Working memory*. Elsevier, 1986, vol. 11.

[15] R. Pienta, M. Kahng, Z. Lin, J. Vreeken, P. Talukdar, J. Abello, G. Parameswaran, and D. H. Chau, "FACETS: Adaptive Local Exploration of Large Graphs," in *Proceedings of the 2017 SIAM International Conference on Data Mining*, SIAM, 2017, pp. 597–605.

[16] P. J. Polack, S.-T. Chen, M. Kahng, M. Sharmin, and D. H. Chau, "TimeStitch: Interactive multi-focus cohort discovery and comparison," in *Visual Analytics Science and Technology (VAST), 2015 IEEE Conference on*, IEEE, 2015, pp. 209–210.

[17] D. Fang, F. Hohman, P. Polack, H. Sarker, M. Kahng, M. Sharmin, M. al'Absi, and D. H. Chau, "mHealth visual discovery dashboard," in *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, ACM, 2017, pp. 237–240.

[18] DARPA, *DARPA-BAA-10-50: Graph Understanding and Analysis for Rapid Detection - Deployed on the Ground (GUARD DOG)*, 2010 (accessed Jan 7, 2017).

[19] A. Endert, M. S. Hossain, N. Ramakrishnan, C. North, P. Fiaux, and C. Andrews, "The human is the loop: new directions for visual analytics," *Journal of intelligent information systems*, vol. 43, no. 3, pp. 411–435, 2014.

[20] F. McSherry, M. Isard, and D. G. Murray, "Scalability! But at what COST?" In *HotOS*, 2015.

[21] Z. Liu and J. Heer, "The effects of interactive latency on exploratory visual analysis," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 2122–2131, 2014.

[22] Z. Lin, M. Kahng, K. M. Sabrin, D. H. P. Chau, H. Lee, and U Kang, "Mmap: Fast billion-scale graph computation on a pc via memory mapping," in *Big Data (Big Data), 2014 IEEE International Conference on*, IEEE, 2014, pp. 159–164.

[23] S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Kaashoek, R. Morris, N. Zeldovich, *et al.*, "An analysis of linux scalability to many cores.," in *OSDI*, vol. 10, 2010, pp. 86–93.

[24] R. R. Curtin, J. R. Cline, N. P. Slagle, W. B. March, P. Ram, N. A. Mehta, and A. G. Gray, "mlpack: A scalable C++ machine learning library," *Journal of Machine Learning Research*, vol. 14, pp. 801–805, 2013.

[25] P. J. Polack, M. Sharmin, K. de Barbaro, M. Kahng, S.-T. Chen, and D. H. Chau, "Exploratory visual analytics of mobile health data: Sensemaking challenges and opportunities," in *Mobile Health: Sensors, Analytic Methods, and Applications*, J. M. Rehg, M. S. A., and S. Kumar, Eds., vol. 1, Springer, 2017, ISBN: 978-3-319-51393-5.

[26] N. Saleheen, A. A. Ali, S. M. Hossain, H. Sarker, S. Chatterjee, B. Marlin, E. Ertin, M. al'Absi, and S. Kumar, "Puffmarker: A multi-sensor approach for pinpointing the timing of first lapse in smoking cessation," in *Ubicomp'15*, ACM, 2015, pp. 999–1010.

[27] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: A novel symbolic representation of time series," *Data Mining and knowledge discovery*, vol. 15, no. 2, pp. 107–144, 2007.

[28] C. Flavián, M. Guinalíu, and R. Gurrea, "The role played by perceived usability, satisfaction and consumer trust on website loyalty," *Information & management*, vol. 43, no. 1, pp. 1–14, 2006.

[29] S. McCanne and V. Jacobson, "The bsd packet filter: A new architecture for user-level packet capture.," in *USENIX winter*, vol. 93, 1993.

[30] B. Gregg, *Ebpf: One small step*, `http://www.brendangregg.com/blog/2015-05-15/ebpf-one-small-step.html`, 2015.

[31] ——, *Strace wow much syscall*, `http://www.brendangregg.com/blog/2014-05-11/strace-wow-much-syscall.html`, (Accessed on 02/16/2018).

[32] B. Gregg and J. Mauro, *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD*. Prentice Hall Professional, 2011.

[33] B. Gregg, *The pmcs of ec2: Measuring ipc*, `http://www.brendangregg.com/blog/2017-05-04/the-pmcs-of-ec2.html`, (Accessed on 02/16/2018).

[34] *Iovisor/bcc: Bcc - tools for bpf-based linux io analysis, networking, monitoring, and more*, `https://github.com/iovisor/bcc`, (Accessed on 02/16/2018).

[35] J. Angel, *Joelagnel/bpfd: Bpfd: Berkeley packet filter daemon (bpfd). makes it possible to run bcc tools across systems.* `https://github.com/joelagnel/bpfd`, (Accessed on 02/16/2018).

[36] S. Goldshtein, *Usdt/bpf tracing tools: Java, python, ruby, node, mysql, postgresql | all your base are belong to us*, `http://blogs.microsoft.co.il/sasha/2016/12/23/usdtbpf-tracing-tools-java-python-ruby-node-mysql-postgresql/`, (Accessed on 02/16/2018).

[37] B. Gregg, "Visualizing system latency," *Communications of the ACM*, vol. 53, no. 7, pp. 48–54, 2010.

[38] *Detecting outliers in cloud infrastructure with datadog heatmaps*, `https://www.datadoghq.com/blog/detecting-outliers-cloud-infrastructure-datadog-heatmaps/`, (Accessed on 02/16/2018).